

# STRUCTURAL OPERATIONAL SEMANTICS

Grigore Rosu

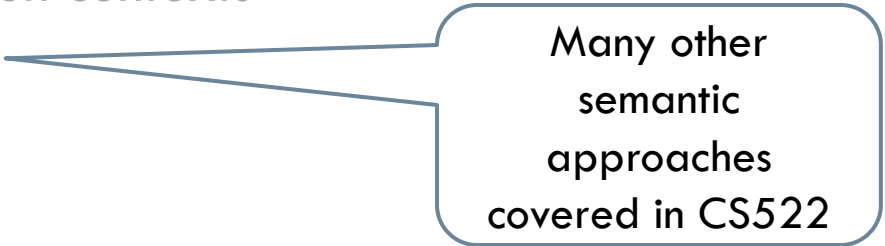
CS422 – Programming Language Semantics

# Conventional Semantic Approaches

2

A language designer should understand the existing design approaches, techniques and tools, to know what is possible and how, or to come up with better ones. This part of the course will cover the two major PL semantic approaches:

- Big-step structural operational semantics (Big-step SOS)
- Small-step structural operational semantics (Small-step SOS)
- Denotational semantics
- Modular structural operational semantics (Modular SOS)
- Reduction semantics with evaluation contexts
- Abstract Machines
- The chemical abstract machine
- Axiomatic semantics



Many other  
semantic  
approaches  
covered in CS522

# IMP

A simple imperative language

# IMP – A Simple Imperative Language

4

We will exemplify the conventional semantic approaches by means of IMP, a very simple non-procedural imperative language, with

- Arithmetic expressions
- Boolean expressions
- Assignment statements
- Conditional statements
- While loop statements
- Blocks

# IMP Syntax

5

*Int* ::= the domain of (unbounded) integer numbers, with usual operations on them  
*Bool* ::= the domain of Booleans  
*Id* ::= standard identifiers  
*AExp* ::= *Int*  
          | *Id*  
          | *AExp* + *AExp*  
          | *AExp* / *AExp*  
*BExp* ::= *Bool*  
          | *AExp* <= *AExp*  
          | ! *BExp*  
          | *BExp* && *BExp*  
*Block* ::= {}  
          | { *Stmt* }  
*Stmt* ::= *Block*  
          | *Id* = *AExp* ;  
          | *Stmt Stmt*  
          | if ( *BExp* ) *Block* else *Block*  
          | while ( *BExp* ) *Block*  
*Pgm* ::= int List { *Id* } ; *Stmt*

Suppose that, for demonstration purposes, we want “+” and “/” to be non-deterministically strict, “<=” to be sequentially strict, and “&&” to be short-circuited

Comma-separated list of identifiers

# IMP State

6

- Most semantics need some notion of *state*. A state holds all the semantic ingredients to fully define the meaning of a given program or fragment of program.
- For IMP, a state is a *partial finite-domain function* from identifiers to integers (i.e., a function defined only on a finite subset of identifiers and undefined on the rest), written using a half-arrow:

$$\sigma : Id \rightarrow Int$$

- We let *State* denote the set of such functions, and may write it

$$[Id \rightarrow Int]^{finite}$$

or

$$\mathbf{Map}\{Id \mapsto Int\}$$

# Lookup, Update and Initialization

7

- We may write states by enumerating each identifier binding.  
For example, the following state binds  $x$  to 8 and  $y$  to 0:

$$\sigma = x \mapsto 8, y \mapsto 0$$

- Typical state operations are lookup, update and initialization

- *Lookup*

$$_(-) : State \times Id \rightarrow Int$$

- *Update*

$$_-[_/_] : State \times Int \times Id \rightarrow State$$

- *Initialization*

$$_ \mapsto _ : \mathbf{List}\{Id\} \times Int \rightarrow State$$

# BIG-STEP SOS

Big-step structural operational semantics

# Big-Step Structural Operational Semantics (Big-Step SOS)

9

- Gilles Kahn (1987), under the name *natural semantics*. Also known as *relational semantics*, or *evaluation semantics*. We can regard a big-step SOS as a recursive interpreter, telling for a fragment of code and state what it evaluates to.
- **Configuration**: tuple containing code and semantic ingredients
  - ▣ E.g.,  $\langle a_1, \sigma \rangle \quad \langle a_1 + a_2, \sigma \rangle \quad \langle i_1 \rangle \quad \langle i_1 +_{Int} i_2 \rangle \quad \langle \sigma \rangle$
- **Sequent**: Pair of configurations, to be *derived* or *proved*
  - ▣ E.g.,  $\langle a_1, \sigma \rangle \Downarrow \langle i_1 \rangle \quad \langle a_1 + a_2, \sigma \rangle \Downarrow \langle i_1 +_{Int} i_2 \rangle$
- **Rule**: Tells how to derive a sequent from others

▣ E.g.,

Premises

$$\frac{\langle a_1, \sigma \rangle \Downarrow \langle i_1 \rangle \quad \langle a_2, \sigma \rangle \Downarrow \langle i_2 \rangle}{\langle a_1 + a_2, \sigma \rangle \Downarrow \langle i_1 +_{Int} i_2 \rangle}$$

Conclusion

$$\langle a_1 + a_2, \sigma \rangle \Downarrow \langle i_1 +_{Int} i_2 \rangle$$

Read  
“evaluates to”

# Big-Step SOS of IMP - Arithmetic

10

$$\langle i, \sigma \rangle \Downarrow \langle i \rangle$$

State  
lookup

(BIGSTEP-INT)

$$\langle x, \sigma \rangle \Downarrow \langle \sigma(x) \rangle \quad \text{if } \sigma(x) \neq \perp$$

(BIGSTEP-LOOKUP)

Read: “provided that  $a_1$  evaluates to  $i_1$  in  $\sigma$  and  $a_2$  evaluates to  $i_2$  in  $\sigma$ , then  $a_1 + a_2$  evaluates to the integer sum of  $i_1$  and  $i_2$  in  $\sigma$ ”

$$\frac{\langle a_1, \sigma \rangle \Downarrow \langle i_1 \rangle \quad \langle a_2, \sigma \rangle \Downarrow \langle i_2 \rangle}{\langle a_1 + a_2, \sigma \rangle \Downarrow \langle i_1 +_{Int} i_2 \rangle}$$

(BIGSTEP-ADD)

$$\frac{\langle a_1, \sigma \rangle \Downarrow \langle i_1 \rangle \quad \langle a_2, \sigma \rangle \Downarrow \langle i_2 \rangle}{\langle a_1 / a_2, \sigma \rangle \Downarrow \langle i_1 /_{Int} i_2 \rangle}$$

if  $i_2 \neq 0$

(BIGSTEP-DIV)

Side condition ensures rule will never apply when  $a_2$  evaluates to 0

# Big-Step SOS of IMP - Boolean

11

$$\langle t, \sigma \rangle \Downarrow \langle t \rangle$$

(BIGSTEP-BOOL)

$$\frac{\langle a_1, \sigma \rangle \Downarrow \langle i_1 \rangle \quad \langle a_2, \sigma \rangle \Downarrow \langle i_2 \rangle}{\langle a_1 \leq a_2, \sigma \rangle \Downarrow \langle i_1 \leq_{Int} i_2 \rangle}$$

(BIGSTEP-LEQ)

$$\frac{\langle b, \sigma \rangle \Downarrow \langle \text{true} \rangle}{\langle ! b, \sigma \rangle \Downarrow \langle \text{false} \rangle}$$

(BIGSTEP-NOT-TRUE)

$$\frac{\langle b, \sigma \rangle \Downarrow \langle \text{false} \rangle}{\langle ! b, \sigma \rangle \Downarrow \langle \text{true} \rangle}$$

(BIGSTEP-NOT-FALSE)

$$\frac{\langle b_1, \sigma \rangle \Downarrow \langle \text{false} \rangle}{\langle b_1 \&\& b_2, \sigma \rangle \Downarrow \langle \text{false} \rangle}$$

(BIGSTEP-AND-FALSE)

$$\frac{\langle b_1, \sigma \rangle \Downarrow \langle \text{true} \rangle \quad \langle b_2, \sigma \rangle \Downarrow \langle t \rangle}{\langle b_1 \&\& b_2, \sigma \rangle \Downarrow \langle t \rangle}$$

(BIGSTEP-AND-TRUE)

# Big-Step SOS of IMP - Statements

12

$$\langle \{\}, \sigma \rangle \Downarrow \langle \sigma \rangle$$

(BIGSTEP-EMPTY-BLOCK)

$$\frac{\langle s, \sigma \rangle \Downarrow \langle \sigma' \rangle}{\langle \{s\}, \sigma \rangle \Downarrow \langle \sigma' \rangle}$$

State  
update

(BIGSTEP-BLOCK)

$$\frac{\langle a, \sigma \rangle \Downarrow \langle i \rangle}{\langle x = a; , \sigma \rangle \Downarrow \langle \sigma[i/x] \rangle} \text{ if } \sigma(x) \neq \perp$$

(BIGSTEP-ASGN)

$$\frac{\langle s_1, \sigma \rangle \Downarrow \langle \sigma_1 \rangle \quad \langle s_2, \sigma_1 \rangle \Downarrow \langle \sigma_2 \rangle}{\langle s_1 \ s_2, \sigma \rangle \Downarrow \langle \sigma_2 \rangle}$$

(BIGSTEP-SEQ)

$$\frac{\langle b, \sigma \rangle \Downarrow \langle \text{true} \rangle \quad \langle s_1, \sigma \rangle \Downarrow \langle \sigma_1 \rangle}{\langle \text{if } (b) \ s_1 \text{ else } s_2, \sigma \rangle \Downarrow \langle \sigma_1 \rangle}$$

(BIGSTEP-IF-TRUE)

$$\frac{\langle b, \sigma \rangle \Downarrow \langle \text{false} \rangle \quad \langle s_2, \sigma \rangle \Downarrow \langle \sigma_2 \rangle}{\langle \text{if } (b) \ s_1 \text{ else } s_2, \sigma \rangle \Downarrow \langle \sigma_2 \rangle}$$

(BIGSTEP-IF-FALSE)

$$\frac{\langle b, \sigma \rangle \Downarrow \langle \text{false} \rangle}{\langle \text{while } (b) \ s, \sigma \rangle \Downarrow \langle \sigma \rangle}$$

(BIGSTEP-WHILE-FALSE)

$$\frac{\langle b, \sigma \rangle \Downarrow \langle \text{true} \rangle \quad \langle s \text{ while } (b) \ s, \sigma \rangle \Downarrow \langle \sigma' \rangle}{\langle \text{while } (b) \ s, \sigma \rangle \Downarrow \langle \sigma' \rangle}$$

(BIGSTEP-WHILE-TRUE)

# Big-Step SOS of IMP - Programs

13

State  
initialization

$$\frac{\langle s, xl \mapsto 0 \rangle \Downarrow \langle \sigma \rangle}{\langle \text{int } xl; s \rangle \Downarrow \langle \sigma \rangle}$$

(BIGSTEP-VAR)

# Big-Step Rule Instances

14

- Rules are schemas, allowing recursively enumerable many instances; side conditions filter out instances

- E.g., these are correct instances of the rule for division

$$\frac{\langle x, (x \mapsto 8, y \mapsto 0) \rangle \Downarrow \langle 8 \rangle \quad \langle 2, (x \mapsto 8, y \mapsto 0) \rangle \Downarrow \langle 2 \rangle}{\langle x/2, (x \mapsto 8, y \mapsto 0) \rangle \Downarrow \langle 4 \rangle}$$

$$\frac{\langle x, (x \mapsto 8, y \mapsto 0) \rangle \Downarrow \langle 8 \rangle \quad \langle 2, (x \mapsto 8, y \mapsto 0) \rangle \Downarrow \langle 4 \rangle}{\langle x/2, (x \mapsto 8, y \mapsto 0) \rangle \Downarrow \langle 2 \rangle}$$

The second may look suspicious, but it is not. Normally, one should never be able to apply it, because one cannot prove its hypotheses

- However, the following is *not* a correct instance (no matter what ? is):

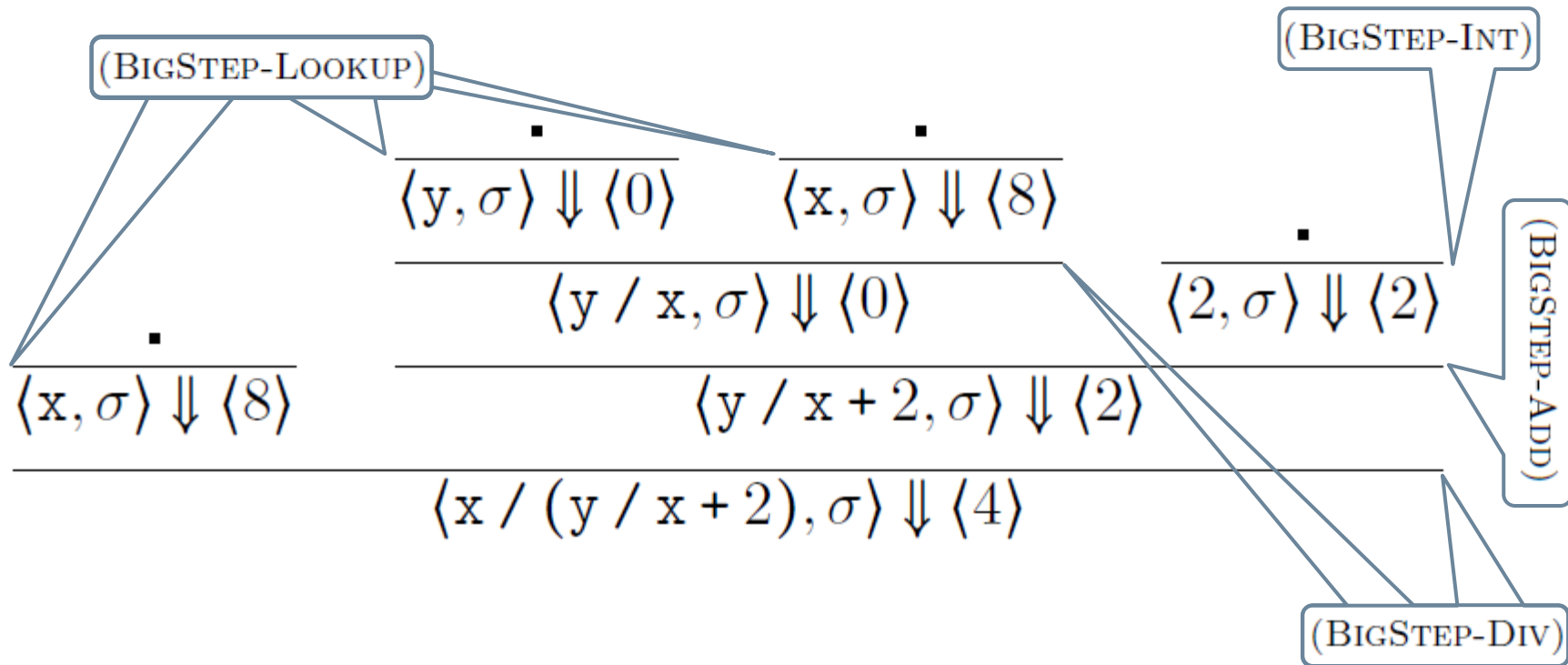
$$\frac{\langle x, (x \mapsto 8, y \mapsto 0) \rangle \Downarrow \langle 8 \rangle \quad \langle y, (x \mapsto 8, y \mapsto 0) \rangle \Downarrow \langle 0 \rangle}{\langle x/y, (x \mapsto 8, y \mapsto 0) \rangle \Downarrow \langle ? \rangle}$$

# Big-Step SOS Derivation

15

The following is a valid proof derivation, or proof tree, using the big-step SOS proof system of IMP above.

Suppose that  $x$  and  $y$  are identifiers and  $\sigma(\mathbf{x})=8$  and  $\sigma(\mathbf{y})=0$ .



# Big-Step SOS for Type Systems

16

- Big-Step SOS is routinely used to define type systems for programming languages
- The idea is that a fragment of code  $c$ , in a given *type environment*  $\Gamma$ , can be assigned a certain type  $\tau$ . We typically write

$$\Gamma \vdash c : \tau$$

instead of

$$\langle c, \Gamma \rangle \Downarrow \langle \tau \rangle$$

- Since all variables in IMP have integer type,  $\Gamma$  can be replaced by a list of untyped variables in our case. In general, however, a type environment  $\Gamma$  contains *typed variables*, that is, pairs “ $x : \tau$ ”.

# Typing Arithmetic Expressions

17

$xl \vdash i : int$  (BIGSTEPTypeSystem-INT)

$xl \vdash x : int$  if  $x \in xl$  (BIGSTEPTypeSystem-LOOKUP)

$$\frac{xl \vdash a_1 : int \quad xl \vdash a_2 : int}{xl \vdash a_1 + a_2 : int}$$
 (BIGSTEPTypeSystem-ADD)

$$\frac{xl \vdash a_1 : int \quad xl \vdash a_2 : int}{xl \vdash a_1 / a_2 : int}$$
 (BIGSTEPTypeSystem-DIV)

$xl \vdash t : bool$  if  $t \in \{\text{true}, \text{false}\}$  (BIGSTEPTypeSystem-BOOL)

# Typing Boolean Expressions

18

$$\frac{xl \vdash a_1 : int \quad xl \vdash a_2 : int}{xl \vdash a_1 \leq a_2 : bool}$$

(BIGSTEPTYPESYSTEM-LEQ)

$$\frac{xl \vdash b : bool}{xl \vdash ! b : bool}$$

(BIGSTEPTYPESYSTEM-NOT)

$$\frac{xl \vdash b_1 : bool \quad xl \vdash b_2 : bool}{xl \vdash b_1 \ \&\& \ b_2 : bool}$$

(BIGSTEPTYPESYSTEM-AND)

$$xl \vdash \{ \} : block$$

(BIGSTEPTYPESYSTEM-EMPTY-BLOCK)

# Typing Statements

19

The type of  $s$   
can be either  
*block* or *stmt*

$$\frac{xl \vdash s : \tau}{xl \vdash \{ s \} : block} \quad \text{if } \tau \in \{block, stmt\}$$

(BIGSTEPTypeSystem-BLOCK)

$$\frac{xl \vdash a : int}{xl \vdash x = a ; : stmt} \quad \text{if } x \in xl$$

(BIGSTEPTypeSystem-ASGN)

$$\frac{xl \vdash s_1 : \tau_1 \quad xl \vdash s_2 : \tau_2}{xl \vdash s_1 s_2 : stmt} \quad \text{if } \tau_1, \tau_2 \in \{block, stmt\}$$

(BIGSTEPTypeSystem-SEQ)

$$\frac{xl \vdash b : bool \quad xl \vdash s_1 : block \quad xl \vdash s_2 : block}{xl \vdash \text{if } (b) s_1 \text{ else } s_2 : stmt}$$

(BIGSTEPTypeSystem-IF)

$$\frac{xl \vdash b : bool \quad xl \vdash s : block}{xl \vdash \text{while } (b) s : stmt}$$

(BIGSTEPTypeSystem-WHILE)

# Typing Programs

20

$$\frac{x\ell \vdash s : \tau}{\vdash \text{int } x\ell ; s : \text{pgm}} \quad \text{if } \tau \in \{block, stmt\} \qquad (\text{BIGSTEP TYPE SYSTEM-PGM})$$

# Big-Step SOS Type Derivation

21

Like the big-step rules for the concrete semantics of IMP, the ones for its type system are also rule schemas. We next show a proof derivation for the well-typed-ness of an IMP program that adds all the numbers from 1 to 100:

$$\frac{\frac{\frac{tree_1}{n, s \vdash (n = 100; s = 0; \text{while } (!(n \leq 0)) \{s = s + n; n = n + -1;\}) : stmt} \quad \frac{\frac{tree_2 \quad tree_3}{n, s \vdash (\text{while } (!(n \leq 0)) \{s = s + n; n = n + -1;\}) : stmt}}{n, s \vdash (n = 100; s = 0; \text{while } (!(n \leq 0)) \{s = s + n; n = n + -1;\}) : stmt}}{\vdash (\text{int } n, s; n = 100; s = 0; \text{while } (!(n \leq 0)) \{s = s + n; n = n + -1;\}) : pgm}$$

where

# Big-Step SOS Type Derivation

22

$$tree_1 = \left\{ \frac{\frac{\overline{\text{n}, s \vdash 100 : int}}{\text{n}, s \vdash (n = 100;) : stmt} \quad \frac{\overline{\text{n}, s \vdash 0 : int}}{\text{n}, s \vdash (s = 0;) : stmt}}{\text{n}, s \vdash (n = 100; s = 0;) : stmt} \right.$$

$$tree_2 = \left\{ \frac{\frac{\overline{\text{n}, s \vdash n : int} \quad \overline{\text{n}, s \vdash 0 : int}}{\text{n}, s \vdash (n \leq 0) : bool}}{\text{n}, s \vdash (! (n \leq 0)) : bool} \right.$$

# Big-Step SOS Type Derivation

23

$$tree_3 = \left\{ \begin{array}{c} \frac{\frac{\frac{\cdot}{n, s \vdash s : int}}{\quad} \quad \frac{\frac{\cdot}{n, s \vdash n : int}}{\quad}}{n, s \vdash (s + n) : int} \quad \frac{\frac{\frac{\cdot}{n, s \vdash n : int}}{\quad} \quad \frac{\frac{\cdot}{n, s \vdash -1 : int}}{\quad}}{n, s \vdash (n + -1) : int} \\ \frac{n, s \vdash (s = s + n;) : stmt \quad n, s \vdash (n = n + -1;) : stmt}{n, s \vdash (s = s + n; n = n + -1;) : stmt} \\ \frac{n, s \vdash (s = s + n; n = n + -1;) : stmt}{n, s \vdash \{ s = s + n; n = n + -1; \} : block} \end{array} \right.$$

# SMALL-STEP SOS

Small-step structural operational semantics

# Small-Step Structural Operational Semantics (Small-Step SOS)

25

- Gordon Plotkin (1981)
- Also known as *transitional semantics*, or *reduction semantics*
- One can regard a small-step SOS as a device capable of executing a program step-by-step
- **Configuration**: tuple containing code and semantic ingredients
  - E.g.,  $\langle a, \sigma \rangle$        $\langle b, \sigma \rangle$        $\langle s, \sigma \rangle$        $\langle p \rangle$
- **Sequent (transition)**: Pair of configurations, to be *derived* (proved)
  - E.g.,  $\langle a_1, \sigma \rangle \rightarrow \langle a'_1, \sigma \rangle$        $\langle a_1 + a_2, \sigma \rangle \rightarrow \langle a'_1 + a_2, \sigma \rangle$
- **Rule**: Tells how to derive a sequent from others

□ E.g.,

$$\frac{\langle a_1, \sigma \rangle \rightarrow \langle a'_1, \sigma \rangle}{\langle a_1 + a_2, \sigma \rangle \rightarrow \langle a'_1 + a_2, \sigma \rangle}$$

# Small-Step SOS of IMP - Arithmetic

26

State  
lookup

$$\langle x, \sigma \rangle \rightarrow \langle \sigma(x), \sigma \rangle \quad \text{if } \sigma(x) \neq \perp \quad (\text{SMALLSTEP-LOOKUP})$$

$$\frac{\langle a_1, \sigma \rangle \rightarrow \langle a'_1, \sigma \rangle}{\langle a_1 + a_2, \sigma \rangle \rightarrow \langle a'_1 + a_2, \sigma \rangle} \quad (\text{SMALLSTEP-ADD-ARG1})$$

$$\frac{\langle a_2, \sigma \rangle \rightarrow \langle a'_2, \sigma \rangle}{\langle a_1 + a_2, \sigma \rangle \rightarrow \langle a_1 + a'_2, \sigma \rangle} \quad (\text{SMALLSTEP-ADD-ARG2})$$

$$\langle i_1 + i_2, \sigma \rangle \rightarrow \langle i_1 +_{Int} i_2, \sigma \rangle \quad (\text{SMALLSTEP-ADD})$$

+ is non-deterministic (its arguments can evaluate in any order, and interleaved)

# Small-Step SOS of IMP - Arithmetic

27

$$\frac{\langle a_1, \sigma \rangle \rightarrow \langle a'_1, \sigma \rangle}{\langle a_1 / a_2, \sigma \rangle \rightarrow \langle a'_1 / a_2, \sigma \rangle} \quad (\text{SMALLSTEP-DIV-ARG1})$$

$$\frac{\langle a_2, \sigma \rangle \rightarrow \langle a'_2, \sigma \rangle}{\langle a_1 / a_2, \sigma \rangle \rightarrow \langle a_1 / a'_2, \sigma \rangle} \quad (\text{SMALLSTEP-DIV-ARG2})$$

$$\langle i_1 / i_2, \sigma \rangle \rightarrow \langle i_1 /_{Int} i_2, \sigma \rangle \quad \text{if } i_2 \neq 0 \quad (\text{SMALLSTEP-DIV})$$

Side condition ensures rule will never apply when denominator is 0

/ is also non-deterministic

# Small-Step SOS of IMP - Boolean

28

$$\frac{\langle a_1, \sigma \rangle \rightarrow \langle a'_1, \sigma \rangle}{\langle a_1 \leq a_2, \sigma \rangle \rightarrow \langle a'_1 \leq a_2, \sigma \rangle} \quad (\text{SMALLSTEP-LEQ-ARG1})$$

$$\frac{\langle a_2, \sigma \rangle \rightarrow \langle a'_2, \sigma \rangle}{\langle i_1 \leq a_2, \sigma \rangle \rightarrow \langle i_1 \leq a'_2, \sigma \rangle} \quad (\text{SMALLSTEP-LEQ-ARG2})$$

$$\langle i_1 \leq i_2, \sigma \rangle \rightarrow \langle i_1 \leq_{Int} i_2, \sigma \rangle \quad (\text{SMALLSTEP-LEQ})$$

Ensures  
sequential  
strictness

# Small-Step SOS of IMP - Boolean

29

$$\frac{\langle b, \sigma \rangle \rightarrow \langle b', \sigma \rangle}{\langle ! b, \sigma \rangle \rightarrow \langle ! b', \sigma \rangle} \quad (\text{SMALLSTEP-NOT-ARG})$$

$$\langle ! \text{true}, \sigma \rangle \rightarrow \langle \text{false}, \sigma \rangle \quad (\text{SMALLSTEP-NOT-TRUE})$$

$$\langle ! \text{false}, \sigma \rangle \rightarrow \langle \text{true}, \sigma \rangle \quad (\text{SMALLSTEP-NOT-FALSE})$$

$$\frac{\langle b_1, \sigma \rangle \rightarrow \langle b'_1, \sigma \rangle}{\langle b_1 \ \&\& \ b_2, \sigma \rangle \rightarrow \langle b'_1 \ \&\& \ b_2, \sigma \rangle} \quad (\text{SMALLSTEP-AND-ARG1})$$

$$\langle \text{false} \ \&\& \ b_2, \sigma \rangle \rightarrow \langle \text{false}, \sigma \rangle \quad (\text{SMALLSTEP-AND-FALSE})$$

$$\langle \text{true} \ \&\& \ b_2, \sigma \rangle \rightarrow \langle b_2, \sigma \rangle \quad (\text{SMALLSTEP-AND-TRUE})$$

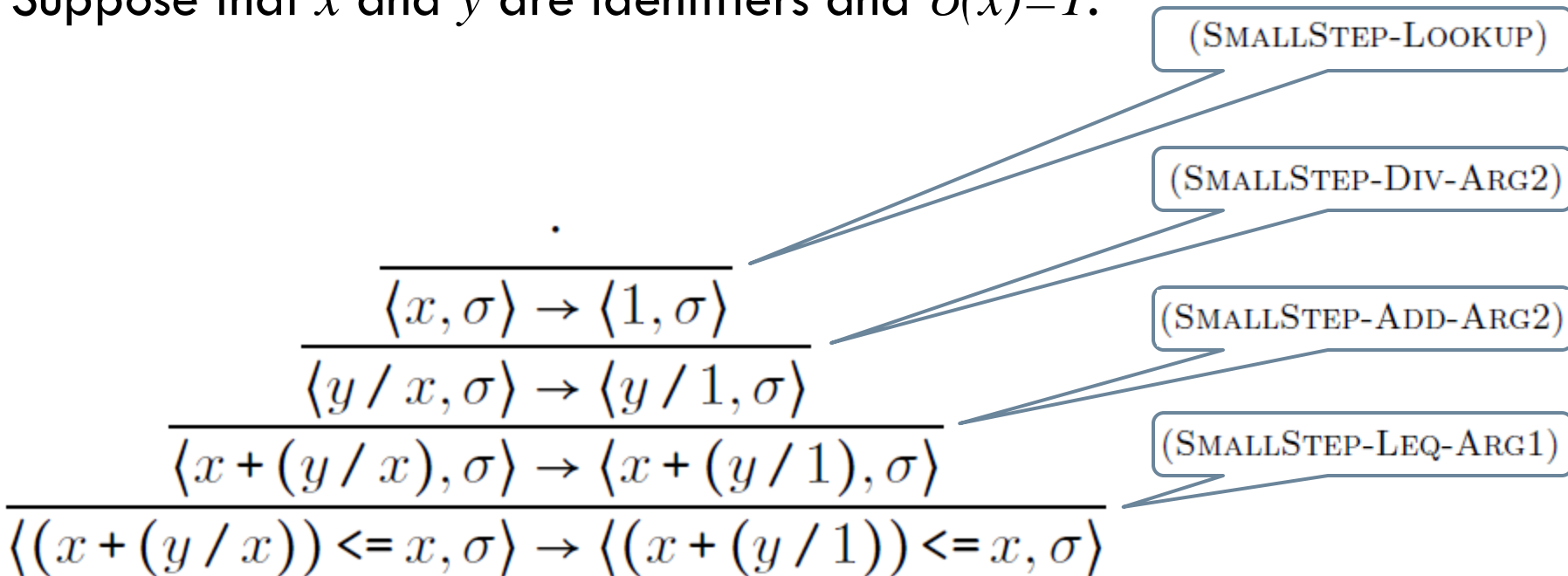
Short-circuit  
semantics

# Small-Step SOS Derivation

30

The following is a valid proof derivation, or proof tree, using the small-step SOS proof system for expressions of IMP above.

Suppose that  $x$  and  $y$  are identifiers and  $\sigma(x)=1$ .



# Small-Step SOS of IMP - Statements

31

$$\langle \{ s \}, \sigma \rangle \rightarrow \langle s, \sigma \rangle$$

(SMALLSTEP-BLOCK)

$$\frac{\langle a, \sigma \rangle \rightarrow \langle a', \sigma \rangle}{\langle x = a;; \sigma \rangle \rightarrow \langle x = a';, \sigma \rangle}$$

State  
update

(SMALLSTEP-ASGN-ARG2)

$$\langle x = i;; \sigma \rangle \rightarrow \langle \{\}, \sigma[i/x] \rangle \quad \text{if } \sigma(x) \neq \perp$$

(SMALLSTEP-ASGN)

$$\frac{\langle s_1, \sigma \rangle \rightarrow \langle s'_1, \sigma' \rangle}{\langle s_1 \ s_2, \sigma \rangle \rightarrow \langle s'_1 \ s_2, \sigma' \rangle}$$

(SMALLSTEP-SEQ-ARG1)

$$\langle \{\} \ s_2, \sigma \rangle \rightarrow \langle s_2, \sigma \rangle$$

(SMALLSTEP-SEQ-EMPTY-BLOCK)

# Small-Step SOS of IMP - Statements

32

$$\frac{\langle b, \sigma \rangle \rightarrow \langle b', \sigma \rangle}{\langle \text{if } (b) \ s_1 \text{ else } s_2, \sigma \rangle \rightarrow \langle \text{if } (b') \ s_1 \text{ else } s_2, \sigma \rangle} \quad (\text{SMALLSTEP-IF-ARG1})$$

$$\langle \text{if } (\text{true}) \ s_1 \text{ else } s_2, \sigma \rangle \rightarrow \langle s_1, \sigma \rangle \quad (\text{SMALLSTEP-IF-TRUE})$$

$$\langle \text{if } (\text{false}) \ s_1 \text{ else } s_2, \sigma \rangle \rightarrow \langle s_2, \sigma \rangle \quad (\text{SMALLSTEP-IF-FALSE})$$

$$\langle \text{while } (b) \ s, \sigma \rangle \rightarrow \langle \text{if } (b) \ \{ s \ \text{while } (b) \ s \} \text{ else } \{\}, \sigma \rangle \quad (\text{SMALLSTEP-WHILE})$$

$$\langle \text{int } xl; s \rangle \rightarrow \langle s, xl \mapsto 0 \rangle \quad (\text{SMALLSTEP-VAR})$$



State  
initialization